Week 5 : Python (continued)

pending tasks

type

Commenting and Error Handling

- Comments are used to add basic documentation to the code.
- Docstrings are used in functions to help in understanding the function.
- A good practice is to avoid adding things precisely said in the code, as natural language is more ambiguous. Do not have redundant comments.

📚 **TextBook : Clean Code - by Robert C. Martin.**

- Runtime errors can be handled using try, except blocks. Multiple exceptions can be defined.

```
# example of commenting and error handling
def input_single_float_inverse(prompt):
    """
    This prompts user to input, and then converts this to a single float inverse
    """
    user_input = input(prompt)
    output = None
    try:
        output = 1/float(user_input)
    except ValueError:
        print("Input value must be a float")
    except ZeroDivisionError:
        print("Input value must not be zero")
    return output
```

- NoneType is the type for the None object, which is an object that indicates no value. None is the return value of functions that "don't return anything".

Lists

- Lists are compound, mutable type : can store data of different types.
- str.split() returns a list of words in the string (space separated words).
- List is represented by , containing the items of the list separated by comma (,).

- Individual items of the list can be accessed using indexing the list using [].
- Lists can be sliced using [start index : end index] inclusive of start index and exclusive of end index.
- Indexing can be done forwards starting from 0 and backwards starting with -1.
- Default start of the list is index 0 and end index is -1.
- List is an iterable.

```
# example of iterating through a list.
fibonacci_list = [1, 1, 2, 3, 5, 8, 13, 21]
for num in fibonacci_list:
    print(num + 10)
```

- List can also be iterated using the number of elements in the list.

```
# example to iterate using the list range function
length = len(fibonacci_list)
for i in range(length):
    print(i, fibonacci_list[i])
```

- New elements can be added to the list using list.append() method.

```
bmi_record[0] = "Krishna A"
bmi_record.append("krishnaa@idontknow.com")
```

Lists - continued

- List of lists : Eg. 2D list, to access the elements use two indices.Each element in the list can thus be updated individually.

```
bmi_dataset = [
               ["Krishna", 75, 1.73, False],
               ["Bheem", 120, 1.78, True]
]
print(type(bmi_dataset[0][0]))
>>> <class 'str'>
```

- Lists have shallow copy.i.e same object will be assigned a new pointer. Thus, updation of any copy will be to the same memory location.
- One of the methods to create a deep copy is the following:

```
A = [1, 2]
B = [item for item in A]
```

Question :

Given the following square matrix A

$$A = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

compute the matrix A^n obtained by multiplying A to itself n times.

Can you comment on the left-top value in A^n (what does this remind you of?)

Tuples and sets

- **()** are used to define a tuple and the elements are accessed like in lists.

```
bmi_categories = ("Underweight", "Normal", "Overweight", "Very overweight")
```

- Tuples are immutable and ordered,this helps to avoid accidental editing of collection objects used in the code.

```
bmi_categories[3] = "VERY overweight"
>>> TypeError: 'tuple' object does not support item assignment
```

- tuple.index('variable_name') returns the index of the variable in the tuple. If the element is not in the tuple, an error is thrown.

```
bmi_categories.index("Normal")
```

- **in** can be used to check if a given element is in the tuple, this will return a boolean value and thus, no errors will be thrown if the element is not in the list.

```
print("Underweight" in bmi_categories)
>>> True
```

- **Magic command :** %%timeit : used to time a particular cell
- Tuples are faster than lists as lists might grow but tuples cannot be updated once created.

```
%%timeit -n1 -r10
for i in range(1000000):
    my_list = ["Tuples", "are", "faster", "than", "lists"]
>>> 1 loop, best of 10: 70.7 ms per loop

%%timeit -n1 -r10
for i in range(1000000):
    my_tuple = ("Tuples", "are", "faster", "than", "lists")
>>> 1 loop, best of 10: 25.1 ms per loop
```

- sets are defined using {}.

```
batsmen = {"Rohit", "Virat", "Ravindra", "Rahul"}
```

- Sets are unordered, thus cannot be indexed, in can be used to check membership.

```
print("Rohit" in batsmen)
>>> True
```

- sets functions: set1.intersection(set2)

```
bowlers = {"Ishant", "Bumrah", "Ravindra"}
all_rounders = batsmen.intersection(bowlers)
print(all_rounders)
>>> {'Ravindra'}
```

- set1.union(set2)

```
players = batsmen.union(bowlers)
print(players)
{'Ravindra', 'Rahul', 'Ishant', 'Bumrah', 'Rohit', 'Virat'}
```

- sets will not store duplicates. It stores only unique elements.
- sets are faster for performing membership operations compared to lists and tuples, as only unique values are stored.

```
%%timeit -n1 -r10
my_list = list(range(10000))
for i in range(10000):
    b = 1947 in my_list
>>> 1 loop, best of 10: 218 ms per loop

%%timeit -n1 -r10
my_tuple = tuple(range(10000))
for i in range(10000):
    b = 1947 in my_tuple
>>> 1 loop, best of 10: 215 ms per loop

%%timeit -n1 -r10
my_set = set(range(10000))
for i in range(10000):
    b = 1947 in my_set
>>> 1 loop, best of 10: 998 µs per loop
```

Dictionaries

- It is saved as key,value pairs
- Defined using **{ 'key' : 'value' }.** , in the {} multiple key value pairs can be stored.

```
bmi_record = {
    "name": "Krishna",
    "weight": 75,
    "height": 1.73,
    "is_overweight": False
}
```

- All keys of a dictionary must be unique.
- The dictionaries can be indexed using the value of the key.

```
bmi_record["name"]
>>> 'Krishna'
bmi_record["email"] = "krishna@idontknow.com"
```

- Iterating through a dict returns the key values.

```
for key in bmi_record:
    print(key, ":", bmi_record[key])
>>> name : Krishna
    weight : 75
    height : 1.73
    is_overweight : False
    email : krishna@idontknow.com
```

- dict.keys() returns all the keys in the dictionary.

```
print(bmi_record.keys())
>>> dict_keys(['name', 'weight', 'height', 'is_overweight', 'email'])
```

- A list of dictionaries can be saved and accessed as follows.

```
bmi_record_1 = {
    "name": "Bheem",
    "weight": 125,
    "height": 1.75,
    "is_overweight": True,
    "email": "bheem@foodtruck.com"
}
bmi_dataset = [bmi_record, bmi_record_1]
print(bmi_dataset[0]["is_overweight"])
>>> False
```

Question :

> Receive the user input that says something like "I went to Parliament" or "Why don't you go to Wankhede". You then subsitute the POI by its lat and long coordinates of that POI.

Question :

Multiple cars are annotated in each image. Each car has a string name followed by a series of points forming a closed polygon. The number of these points varies from car to car. Each point itself is characterised by two numbers corresponding to the x and y coordinates. What data structures would you use to represent this. Note that you have multiple levels of representation that are nested and different choices can be made for each level.

File Handling - Read

- The interpreter has to send a request to the OS to open a file in the file system, this returns a file object.
- The entire file can be read at one go , or read line by line.

```
f = open("poem.txt", "r")
# reading the file at one go
output = f.read()
# reading line from the file
f = open("poem.txt", "r")
out_line = f.readline()
print(out_line)
# reading all lines in a file into a list of lines
out_lines = f.readlines()
print(out_lines)

f.close()
```

- Each line can be read into a list.
- Files opened have to be closed as this might restrict the access to files by other applications. Closing files also releases memory.
- To avoid explicitly closing the files, the following logic can be used:

```
with open("poem.txt", "r") as f:
    out_lines = f.readlines()
```

File Handling - Write

COMMENT - could use enumerate for indexing the lines

- The writelines is complete only after the file object is closed. Until then the text remains in python's buffer.

```
# open a file to read contents into a list
f = open("poem.txt", "r")
out_lines = f.readlines()
f.close()
# method1 to write into a file
f_out = open("fav.txt", "w")
f_out.writelines(out_lines[2])
f_out.close()
# method2 to write into a file
fav_lines = [2, 22]
with open("fav.txt", "w") as f_out:
    for fav_line in fav_lines:
        f_out.writelines(out_lines[fav_line])
```

Question:

1. Create a dictionary whose keys correspond to unique words in a given file and whose values correspond to the number of times each word appears in the file.
2. To an output file, write the 10 most frequent words in separate lines.
3. To an output file, write the 5 most frequenct 2-grams in separate lines.

MCQ : Week 5

1. What is the output of the following code?

```
x = [10, 20, 30, 40]
del x[0:6]
print(x)
```

    1. []
    2. **List index out of range.**
    3. [10,20]

2. Select all the correct options to copy a list
    1. newList = copy(aList)
    2. **newList = aList.copy()**
    3. **newList.copy(aList)**
    4. newList = list(aList)
3. Which of the following will give output as [23,2,9,75] ?
    1. print(list1[1:7:2])
    2. print(list1[0:7:2])
    3. **print(list1[1:8:2])**
    4. print(list1[0:8:2])
4. Select the true statement
    1. Both tuples and lists are immutable.
    2. **Tuples are immutable while lists are mutable.**
    3. Both tuples and lists are mutable.
    4. Tuples are mutable while lists are immutable.
5. Which of the following will not throw an error ?

```
tupl=(5,2,7,0,3)
```

    1. tupl[1]=2
    2. tupl.append(2)
    3. **tupl1 = tupl+tupl**
    4. tupl.sort()
6. Which of the following statements is used to create an empty set?
    1. {}
    2. **set()**
    3. []
    4. ()
7. What is the output of the following code?

```
sampleSet = {"Yellow", "Orange", "Black"}
print(sampleSet[1])
```

1. Yellow
2. **TypeError: 'set' object is not subscriptable**
3. Orange

8. What is the output of the following code?

```
d1={"abc":5,"def":6,"ghi":7}
print(d1[0])
```

1. abc
2. 5
3. {"abc":5}
4. **Error**

9. Which of the following commands can be used to read “n” characters from a file using the file object ‘file’?
    1. file.read(n)
    2. n = file.read()
    3. file.readline(n)
    4. file.readlines()
10. Which of the following attributes related to a file object?
    1. **close**
    2. **mode**
    3. **name**
    4. rename